ANSWERS

A6.10 Project: the mission robot

Theory of computation · A level · AQA 7517 4.4.2.1 · about 30 min

What this lesson is about

Validate a mission with a regular expression, carry it out with a Mealy machine, and report with sets.

- The mission language
- The controller
- What the robot saw
- Where the limits are

Questions

5 marks in all

1. A mission is one or more ids separated by commas, defined by <mission> ::= <id> | <id>,<mission>. Why is a regular expression enough to check it? [1 mark]

- ☐ **A** The rules only recurse at the end, so nothing needs to be counted or matched: the language is regular
- ☐ **B** Regular expressions can check any BNF grammar
- ☐ **C** It has fewer than ten rules
- ☐ **D** It uses digits

> **Answer:** A. Recursion only at the end is repetition, which a regex can express. Nesting in the middle of a rule would need BNF.

2. Which strings are matched in full by [0-9]+(,[0-9]+)*? [1 mark]

Tick every answer that is true.

- ☐ **A** 3,8,5
- ☐ **B** 12
- ☐ **C** 3,,8
- ☐ **D** 3,8,
- ☐ **E** 007

> **Answer:** A, B, E. Each id is one or more digits and each comma must be followed by another id.

3. The robot saw markers {3, 5, 6, 8}. The mission was {3, 5, 8}. What is seen \ mission? Give the member. [1 mark]

> **Answer:** 6. The difference keeps members of seen that are not in the mission.

4. The mission names a marker that is not on the mat, and the controller stays in SEARCH forever. What is the practical answer? [1 mark]

- ☐ **A** Write a checker that detects every possible infinite loop in advance
- ☐ **B** Give the state a time limit and treat running out as a failure
- ☐ **C** Use a faster processor
- ☐ **D** Remove the SEARCH state

> **Answer:** B. The Halting problem means no checker can catch every infinite loop, so real systems use timeouts.

5. The robot is asked to visit 20 markers in whatever order is shortest. Which approach is most sensible? [1 mark]

- ☐ **A** Try every order, since the problem is tractable
- ☐ **B** Use a heuristic such as nearest neighbour, since the problem is intractable
- ☐ **C** Give up, since the problem is non-computable
- ☐ **D** Use a regular expression

> **Answer:** B. This is the travelling salesman problem. 20! orders is far too many, but a heuristic gives a good route quickly.

The task: the mission

Carry out a mission typed in by the user. - Ask for the mission with `input("Mission? ")`. A valid mission matches `[0-9]+(,[0-9]+)*` exactly (check with `re.fullmatch`). While it is not valid, print `invalid mission` and ask again. The input box holds `3,,8` then `3,8,5`, so `invalid mission` is printed exactly once. - For each id in the mission, in order, run the Mealy machine with a table dictionary like the one in lesson A6.2: keys `(state, input)`, values `(next_state, output)`, start state `"SEARCH"`, stopping at `"STOP"`. Each tick: sense, look up, act, change state, `wait(0.1)`. When it reaches STOP, print `reached` and the id, for example `reached 3`. - `sense(target)` and `act(output, steer)` are written for you. `sense` also returns, as a third value, the list of every marker id in view on that tick. - Keep a set of every marker id seen during the whole run. At the end, print `seen but not in mission:` followed by the ids in the set difference, ascending, separated by single spaces, or `none` if it is empty. - Do not touch any marker.

```
# the two lines every program starts with: the commands, then the robot
from bugbot import *
connect()

import re

def bearing_of(cx):
    return (cx - 160) * 120 / 320      # pixels to degrees

def sense(target):
    tags = apriltags()
    ids = [t[0] for t in tags]
    mine = [t for t in tags if t[0] == target]
    if not mine:
        return "none", 0, ids
    if mine[0][3] <= 15:
        return "near", 0, ids
    return "far", bearing_of(mine[0][1]), ids

def act(output, steer):
    if output == "spin":
        turn_right(40)
    elif output == "drive":
        drive(60, 0, steer * 3)
    elif output == "halt":
        stop()

set_cv("apriltag")
```

The hint students can ask for: Do it in stages and test each: first the loop that keeps asking until the mission is valid; then one visit with the machine; then a loop over the ids. Add every id from each sense result to a set as you go, and at the end take away the set of mission ids.

A solution

```
# the two lines every program starts with: the commands, then the robot
from bugbot import *
connect()

import re

def bearing_of(cx):
    return (cx - 160) * 120 / 320      # pixels to degrees
```

```
def sense(target):
    tags = apriltags()
    ids = [t[0] for t in tags]
    mine = [t for t in tags if t[0] == target]
    if not mine:
        return "none", 0, ids
    if mine[0][3] <= 15:
        return "near", 0, ids
    return "far", bearing_of(mine[0][1]), ids

def act(output, steer):
    if output == "spin":
        turn_right(40)
    elif output == "drive":
        drive(60, 0, steer * 3)
    elif output == "halt":
        stop()

set_cv("apriltag")
table = {
    ("SEARCH", "none"): ("SEARCH", "spin"),
    ("SEARCH", "far"): ("APPROACH", "drive"),
    ("SEARCH", "near"): ("STOP", "halt"),
    ("APPROACH", "none"): ("SEARCH", "spin"),
    ("APPROACH", "far"): ("APPROACH", "drive"),
    ("APPROACH", "near"): ("STOP", "halt"),
}
mission = input("Mission? ")
while not re.fullmatch(r"[0-9]+(,[0-9]+)*", mission):
    print("invalid mission")
    mission = input("Mission? ")
ids = [int(part) for part in mission.split(",")]
seen = set()
for target in ids:
    state = "SEARCH"
    while state != "STOP":
        symbol, steer, in_view = sense(target)
        seen.update(in_view)
        state, output = table[(state, symbol)]
        act(output, steer)
        wait(0.1)
    print("reached", target)
extra = seen - set(ids)
print("seen but not in mission:", " ".join(str(x) for x in sorted(extra)) if extra else
"none")
```

Any program that meets the task's checks is marked correct in the simulator; this is one way, not the only way.